P04 Exceptional Vending Machine

Overview

In this assignment, we are going to implement an expanded version of the p01 Vending Machine program. This new version involves the use of instantiable classes and exception handling. The version of the p01 vending machine program used procedural programming (the vending machine represented by an oversize array was passed as input to all the static methods). Also, you might have noticed that the p01 program was not robust. The program crashes in the case of bad input passed to some methods (for instance an item expiration date not parsable to an integer). We'll try to cope with these limitations in this new version of our program. The following new features will be added to the vending machine program.

- Throwing exceptions to report bugs or misuse of methods or operations.
- Loading a set of new items to the vending machine from a file.
- Saving the summary of the vending machine in a file.
- Develop unit tests to check the correctness of the implementation of the different operations supported by our exceptional vending machine.

Learning Objectives

The goals of this assignment include:

- Learn how to improve the robustness of a program so it can survive unusual circumstances and cope with erroneous input without crashing.
- Develop your understanding of the difference between checked and unchecked exceptions, and get practice both throwing and catching exceptions of each kind.
- Get more practice writing instantiable classes.
- Get more practice writing tests, specifically tests that detect whether exceptions are thrown under the prescribed circumstances or not.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P4 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 10/09/2022**. You CANNOT take the pre-assignment quiz for credit passing its deadline. But you can still access it. The pre-assignment quiz contains hints which can help you in the development of this assignment.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Additional Assignment Requirements and Notes

(Please read carefully!)

- Pair programming is **ALLOWED** but not required for this assignment. If you decide to work with a partner on this assignment, REGISTER your partnership NO LATER than **11:59PM on Sunday 10/09/2022** and MAKE SURE that you have read and understood the CS300 Pair Programming Policy
- You are **ALLOWED to import relevant exceptions** in all your submitted source files.
- **You are NOT ALLOWED to import or use** java.util.Arrays or java.util.ArrayList or java.util.List or any other kind of external data structures in all your submitted source files.
- **The ONLY external libraries** you may use in your `ExceptionalVendingMachine` class are defined by these import statements.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.zip.DataFormatException;
import java.util.Scanner;
```

- Only your submitted `ExceptionalVendingMachineTester` class can contain a **main** method.
- You are NOT allowed to add any constant or variable not defined in this write-up **outside of any method**.
- You CAN define local variables (inside a method) that you may need to implement the methods defined in this program.
- You CAN define **private** helper methods to help implement the different public methods (static or instance) defined in this write-up if needed.
- You MUST NOT add any **public methods** to your `Item` and `ExceptionalVendingMachine` classes other than those defined in this write-up.
- All your test methods should be defined and implemented in your `ExceptionalVendingMachineTester.java` file.
- All your test methods must be **public static**. Also, they must take **zero arguments**, **return a boolean**.
- All String comparisons in this assignment are CASE SENSITIVE.
- If you are not familiar with handling exceptions while reading from or writing in files, the zyBook section 3.5 includes some details.
- Feel free to **reuse** any of the provided source code or javadoc comments in this write-up verbatim in your own submission.
- All implemented methods including the main method MUST have their own javadoc-style method headers, with accordance to the CS300 Course Style Guide.
- All your classes MUST have a javadoc-style class header.
- All your data fields defined outside of any method MUST be commented.

- Make sure to submit your code (work in progress) of this assignment on Gradescope both early and often. This will 1) give you time before the deadline to fix any defects that are detected by the tests, 2) provide you with an additional backup of your work, and 3) help track your progress through the implementation of the assignment. These tests are designed to detect and provide feedback about only very specific kinds of defects. **It is your responsibility to implement additional testing to verify that the rest of your code is functioning in accordance with this write-up.**
- You can submit your work in progress multiple times on gradescope. Your submission may include methods not implemented or with partial implementation or with a default return statement. But avoid submitting a code which does not compile. A submission which contains compile errors won't pass any of the automated tests on gradescope.
- You are also responsible for maintaining secure back-ups of your progress as you work. The OneDrive and GoogleDrive accounts associated with your UW NetID are often convenient and secure places to store such backups.
- You MUST adhere to the Academic Conduct Expectations and Advice

1 Getting Started

- Start by creating a new Java Project in eclipse called P04 Exceptional Vending Machine, for instance. As usual, you MUST ensure that your new project uses Java 17, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-17" within the new Java Project dialog box. DO NOT create any module within to your java project. All your source files must be created or added directly to the default source package or your java project. No `package` statements are allowed at the top of your submitted source files.
- Then, create a new class called `Item` and add it to the default package source folder of your project. The `Item` class MUST NOT contain a `main` method.
- Next, download the following source files and add them to the default package source folder of your project.
  - ExceptionalVendingMachine.java file
  - ExceptionalVendingMachineTester.java file
- You might have some compiler errors rising up in the `ExceptionalVendingMachine.java` source file. Disregard them at this level. These errors should be resolved after completing the next step and implementing the class `Item`.
- Details related to the specification of these classes are provided in these javadocs.

2 Implement and test the Item Class

- The class Item models an item to be dispensed by a vending machine. After creating the class `Item`, the first step is to define its data fields. The only fields in this class should be:
  - `description` of type `String`: a **private instance** data field which represents a human readable description of the item.
  - `expirationDate` of type int: a **private instance** data field representing the expiration date of the item, starting at day 0, which represents Jan 1, 2023.
- Then, you should add and implement the constructor of this class. Only one constructor for this class with the signature `public Item(String description, int expirationDate)` should be defined. The specification of this constructor and details related to the exceptions to be thrown in the case of passing invalid inputs are provided in the javadocs of the class Item.
- Read carefully through the details provided in the javadocs of the class `Item` to get a better understanding of the expected behavior of the constructor and every instance method defined in this class. Then, we recommend implementing the following tester methods first. The signatures and the javadoc style method headers of these testers are provided in the starter source file of the ExceptionalVendingMachineTester class.

```
public static boolean testItemConstructorGettersSetters(){}
public static boolean testItemConstructorNotValidInput() {}
public static boolean testItemEquals() {}
```

- Next, implement the constructor, the getter and setter methods, `toString()`, and finally the `equals()` method.
- Notice that the `toString()` and the `equals()` methods override the default behaviors defined in the java.lang.Object class. You MUST add an `@Override` annotation above their method signatures. You MUST also provide a complete javadoc style method header for each of these methods. You should NOT refer the user to the default implementation via the `@see` tag.
- The String representation of an item having a given `description` and a given `expirationDate` must be formatted as follows.

```
description: expirationDate
```

- You can use the `instanceof` operator to check whether the reference passed as input to the method `Item.equals()` references an object instance of the class `Item`.

- The reference passed as input to the `equals` method must be appropriately downcast to the type `Item` to access its instance data fields.
- Recall that we consider CASE SENSITIVE string comparisons in this program.

3 Expand and test the ExceptionalVendingMachine Class

Now, we invite you to read CAREFULLY through the provided starter code of the `ExceptionalVendingMachine` class and the details of its javadocs.

- Notice carefully the changes made to switch the `VendingMachine` class defined in p01 to this instantiable class. You can be able to create multiple instances of exceptional vending machines. Each of them has its own data (the array `items` and its `size`). All methods defined in this class are now `instance` methods. We NO longer need to return the changes made to the size of the vending machine after adding or remove items. The `size` is defined as an instance data field and not an input parameter passed to static methods. All the changes to the size will be maintained after the instance methods return.
- After enjoying reading through the provided starter code and the javadoc comments of the ExceptionalVendingMachine, you can implement the following tester methods.

```
testExceptionalVendingMachineConstructor() {}
public static boolean testExceptionalVendingMachineAddContainsRemoveGetters(){}
public static boolean testEmptySizeFullExceptionalVendingMachine() {}
```

- Next, complete the implementation of the constructor and the following instance methods with accordance to their specification. Your implementation should pass your own tester methods. Your tester methods should also be able to detect potential bugs in any broken implementation, not necessarily yours.

```
public ExceptionalVendingMachine(int capacity){}
public boolean isEmpty() {}
public boolean isFull() {}
public int size() {}
public void addItem(String description, int expirationDate) {}
public String getItemAtIndex(int index) {}
public int getItemOccurrences(String description) {}
public boolean containsItem(String description) {}
public int getItemOccurrencesByExpirationDate(String description, int expirationDate) {}
public int getIndexNextItem(String description) {}
```

- You DO NOT need to make any changes to the implementation of the `getItemsSummary()`

3.1 Implement and test the loadOneItem() method

- Next, implement the `loadOneItem()` method and its tester `testLoadOneItem()` with accordance to the details provided in their javadocs.
- Note that the string *itemRepresentation* passed as input to the `loadOneItem` method should be formatted as follows. Otherwise it will be invalid.

```
<itemDescription>: <itemExpirationDate>
```

- We consider one `colon :` as a delimiter for tokens in the above line.
- You should disregard extra whitespaces at the beginning and end of the item description or the item's expiration time parts.
- The item description should not be blank.
- The item expiration date should be parsable to a positive integer.
- The following methods defined in the java.lang.String class can help you parsing the `itemRepresentation` string and extract its parts.
  - String.trim()
  - String.split()

3.2 Implement and test the loadItems() and saveVendingMachineSummary() methods

- Now, you can implement the `loadItems()` and the `saveVendingMachineSummary()` with respect to the details provided in their javadoc style comments.
- Each of these methods take a java.io.File as input.
- We highly recommend, but we don't require implementing tester methods for the `loadItems` and the `saveVendingMachineSummary` methods. You can name them at your convenience.
- In these tester methods, you can create the file object using the constructor of the java.io.File class that takes a path name String as input parameter.
- You can find here a sample of text file named items.txt which can be used for testing your `loadItems` method. It contains blank lines, correct lines, and badly formatted lines.
- Note that you can use a java.util.Scanner to read the java.io.File object.

- You can use an object of type PrintWriter or FileWriter to write the summary of the vending machine in the provided `java.io.File` object.
- Your *saveVendingMachineSummary()* method should not throw any exception. If an IOException may be thrown in the *saveVendingMachineSummary()*, you MUST catch it.
- Each of your `loadItems()` and `saveVendingMachineSummary()` method MUST contain a `finally` block where the scanner or the file writer should be closed.
- If you are not familiar with file input/output instructions, the details provided in the zyBook section 3.5 can be helpful.

4 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only THREE files that you must submit include: `Item.java`, `ExceptionalVendingMachine.java`, and `ExceptionalVendingMachineTester.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.